10 | 按指定顺序给词语排序，提高查找效率

2021-03-04 尹会生

Python自动化办公实战课 进入课程 >

讲述：尹会生
时长 19:46 大小 18.11M

你好，我是尹会生。

之前我在游戏行业工作的时候，经常需要取得用户在线时长 TOP3、用户战斗力 TOP5、用户完成任务数量 TOP10 等数据，相信你在工作中也会有遇到从大量数据取得 TopN 这类需求。

提取 TopN 的操作，本质上是对大量数据先进行排序，然后根据排序的结果取出前 N 个值。而实现提取 TopN 的值，用 Python 来实现会非常高效，这节课，我就来讲一讲具体怎么操作。

使用 sorted() 函数实现排序

在 Python 中，已经内置了排序函数 sorted() 函数，它是 Python 中实现排序最简单且最直接的形式，可以解决 80% 的排序问题。那么，我们就来学习一下怎么用 sorted() 函数对常见的数据类型进行排序。

先看 sorted() 函数的定义。

```
1 sorted(iterable, cmp=None, key=None, reverse=False)
```

复制代码

sorted 函数共有四个参数，第一个参数是要排序的对象，后面三个参数是排序的方式。

如果要为某个对象排序的话，你可以直接将它作为 sorted() 函数的第一个参数，返回结果会将对象的值进行从小到大的排序。

如果 sorted() 返回的排序结果不满足你的需要，比如你想要从大到小的排序，那你就可以利用后面三个参数来改变排序的方式，实现自定义排序。

所以今天这节课，我会带你学习怎么使用 sorted() 函数实现默认排序和自定义排序，来解决你实际工作中遇到的多种排序问题。我们先来看怎么使用 sorted() 实现默认排序。

默认排序

sorted() 函数的默认排序是按照从小到大的顺序进行排序的。例如一家公司即将上市，需要在职的前 1000 名员工的工号和姓名，工号越小的员工配股越多。这时候就需要给所有员工的工号进行排序，并取得工号排在前 1000 个员工的名字。具体怎么做呢？

我们来看 sorted() 函数的用法。我们从它的第一个参数开始学习。第一个参数是 sorted() 函数要排序的对象，我以列表为例，把员工的工号放到列表中，我们来看一下 sorted() 函数是怎么对列表中的工号进行从小到大的排序的。

```
1 sorted_list = sorted([30, 50, 20, 10, 40])
2 print(sorted_list)
3 # [10, 20, 30, 40, 50]
```

复制代码

我在代码中给 sorted() 函数指定了一个列表类型的参数“[30, 50, 20, 10, 40]”。按照 sorted() 函数的定义，这个列表会被 sorted() 函数按照数字从小到大进行排序，并返回排序后的结果。并且，我把结果放入 sorted_list 对象中，你从代码中可以看到 sorted_list 已经把数据按照默认从小到大的顺序排列好，并把结果放在一个列表数据类型中。

这就是 sorted() 的基本用法，其实是比较简单的，不过这里**有三点需要你注意。**

首先，sorted() 函数的默认排序方式是从小到大进行排序，那么对于列表中的数字就会按照默认的算法进行大小比较。但是除了数字外，列表中还会出现字母或字母与数字混合的方式，你也需要掌握这两种方式的排序处理方法。

比如说英文字母，会按照列表中字符串的第一个字母从 A 到 Z、从 a 到 z 进行排序。如果字符串的第一个字母相同，就会比较第二个字母，以此类推，直到比较完字符串中的所有字符。

由于默认排序的时候不支持同时对英文字母和数字进行排序，那么如果列表中既包含了字母，又包含了数字，在默认排序时程序就会提示一个异常，那就是无法使用“<”比较整数和字符串。

复制代码

```
TypeError: '<' not supported between instances of 'int' and 'str'
```

因此，当列表中出现类似“["a", 1, "bb"]”这种，**既包含字符串又包含数字的列表，进行排序操作的时候**，你需要先统一类型，把数字使用 str() 函数转换为字符串类型，再对列表进行排序。我把代码写出来，供你参考：

复制代码

```
sorted_list2 = sorted(["a", str(1), "bb"])
print(sorted_list2)
```

其次，你需要注意，sorted() 函数不会对原有的列表进行修改，它会把排序好的结果存入到一个新的列表当中。

很多人刚开始使用 Python 时，会把 sorted() 函数和列表自带的函数 sort() 函数混淆。

它们虽然名字很像，但支持的数据类型、调用的方式以及返回的结果都不同。

要记住，sort() 函数是列表数据类型自带的，所以只能对列表数据类型进行排序，不能对其他数据类型排序，但 sorted() 函数可以支持任何可迭代的对象。

在调用方式上，sort() 函数使用的方式是“列表.sort()”格式，这也是调用时和 sorted() 函数的差别。

还有一个最大的差别是，sort() 函数会直接修改当前列表（这种修改称作“原地修改”），并返回一个空值 None。而 **sorted() 函数不会对原有的列表进行修改，它会把排序好的结果存入到一个新的列表当中**。

为了让你更好地区分 sorted() 函数和 sort() 函数，我把 sort() 函数的执行结果也提供给你做参考。

复制代码

```
list3 = ["a", "c", "bb"]
no_value = list3.sort()
print(list3)
# ['a', 'bb', 'c']  执行结果
print(no_value)
# None  执行结果
```

在代码的第二行，是 sort() 函数的调用格式。在第三行，我输出了排序之后的列表，并对列表的值进行了重新排序。在第五行，我们可以看到 no_value 变量的返回值为 None，也就是 sort() 函数会对列表进行原地修改，并使用 None 作为返回。

最后，你需要注意 sorted() 函数能够支持的数据类型非常多，既能支持基础数据类型，又能支持 Python 自带的内置函数。根据它的函数定义，第一个参数是“iterable”对象，表示只要该对象可迭代，sorted() 函数就能对它进行排序。

在基本数据类型中，**序列和映射都可以迭代，**序列是指列表、元组、字符串这三种基本数据类型的总称，映射就是我们使用过的字典。除了基本数据类型之外，像是 range() 、map()、zip() 等内置函数都是可迭代对象，因此掌握 sorted() 函数，可以对以上多种类型和函数进行排序。

掌握了这三点注意事项，在使用 sorted() 为可迭代类型进行排序时，就不会出现什么问题了。那接下来，我来带你学习一下它的自定义排序功能，通过自定义排序，可以让 sorted() 基于不同的数据类型实现更加灵活的排序。

自定义排序

自定义排序是在基本排序基础上，能够支持**更多的排序方式和更复杂的数据类型。比如说：**

> 在排序方式上，我们通过参数，可以把默认的从小到大的排序改为从大到小。
>
> 在对数据类型的支持上，像列表中包含元组这种复杂数据类型，可以通过指定元组的任意一列进行排序。

我们先学习自定义排序是怎么支持更多的排序方式的，它的前提是要先更改默认的排序方式。

例如我想要实现列表的从大到小的排序，并提取 Top 3 这一需求，我首先需要使用函数的第四个参数 reverse 参数改变默认的排序方式，排序后我需要使用列表的切片方式提取前三个元素。

我先把代码写出来，然后再给你详细解释。可以看到，我在代码中使用了 sorted_list4 作为排序后的结果，并对 sorted_list4 进行切片，从而得到指定下标对应的值。

复制代码

```
sorted_list4 = sorted([30, 50, 20, 10, 40], reverse=True)
print(sorted_list4[:3])
# [50, 40, 30]
```

在代码中，我为 sorted() 函数增加了参数 reverse。reverse 参数在定义的时候是 sorted() 函数的第四个参数，由于 sorted() 函数的第二、三个参数我希望保持默认，不需要在调用的时候传入，因此 reverse 参数在调用时就要作为第二个参数来使用。

由于在调用时它的位置和定义的位置不同，那么我必须使用关键字参数“reverse=True”的形式，把 reverse 指定为 sorted() 函数的第二个参数。

增加 reverse 参数之后，sorted() 执行的结果也和默认结果不同，sorted 会将默认排序的结果进行反转。而当五个数字的排序结果反转后，也就以从大到小的方式进行了输出。

你看，利用 sorted() 函数的 reverse 参数就能实现 TopN 场景的排序工作，工作中往往需要只得到 TopN 的结果，不需要将全部的排序结果进行输出，这时候你可以将列表按照你的工作需要提取其中的某一部分，这种操作也被称作列**表的切片。**

假如你想取得 sorted_list4 列表的 Top3 元素，可以对列表排序后，使用 sorted_list4[0:3]的写法提取列表前三个值，这种写法的“0” “3”表示列表的下标， “:”表示获取下标的范围。因为列表的下标从 0 开始，因此要写作[0:3]的形式，也可以使用它的省略形式[:3]，这样列表切片操作就会从 0 开始取值。

总结来说，通过 reverse 参数和列表切片，我们可以在默认排序的基础上，实现基于列表的 TopN 场景下的排序，以及 TopN 结果的提取。

不过现实场景往往都是比较复杂的，要排序的数据类型除了列表中包含数字和字符串外，你最经常见到的还有对列表中包含元组这种复合类型进行排序，以及对字典的键或值进行排序。接下来我来带你学习一下两种主要类型，一种是列表 + 元祖的类型，一种是字典类型。

列表 + 元组的排序方式

列表 + 元组形成的组合数据类型，适合存放包含多个属性的对象。我给你看个例子，

“姓名、性别、身份证号”和“学校、姓名、学号”这两组数据就适合用元组存储。因为元组存储的值是不可变的，而这些数据填入之后一般不会修改，刚好和元组的特性吻合。

而且，这些用户信息在工作中往往会被大量存储在一起，为了便于对它们进行排序和查找。你还需要将多个元组再保存到列表当中，这样就形成了列表 + 元组这种组合类型。

对列表 + 元组的形式进行排序，就需要用自定义排序的字段来实现。那就是在默认排序的基础上，增加 key 参数，并通过 lambda 表达式取得元组第三个位置的学号。

lambda 表达式是简化自定义函数的定义和调用而使用的一种语法，使用 lambda 表达式取得学号的字符串之后，sortd() 就可以实现对元组按照学号进行排序了。

在这里，key 没法通过下标对元组排序，而必须通过函数取得参数具体的值。这是 sorted() 函数为了更灵活地实现排序功能，把设置排序关键字这一功能全部开放给用户造成的。事实上，这也是一种典型的通过牺牲易用性来增加灵活性的设计模式。

这一知识点比较难理解，不过不用担心，我在后续课程陆续为你讲解这类函数，直到你能熟练使用它们为止。

到这里，我们就了解了 sorted() 函数的 key 参数必须使用函数做参数的原因，那就继续来看 **key 参数的 lambda 表达式是怎么简化自定义函数的**。

比如我想取得元组中包含“Jerry”的学号“1003”，以及包含“Tom”的元组学号的“1005”，可以使用这样一段代码来实现函数定义和调用方式：

复制代码

```
1 def s(my_tuple):
2     return my_tuple[2]
3 sorted(students, key=s)
4
5 sorted(students, key=lambda s: s[2])
```

在代码的 1-3 行、第 5 行，分别是通过自定义函数和 lambda 表达式实现的提取元组第三个参数的功能。对比来看，自定义函数的定义要比 lambda 表达式复杂。自定义函数的定义和调用部分在 Python 中要分开编写，这也是简单函数更适合用 lambda 表达式编写的原因。

lambda 表达式通常在函数只有一行语句，且不需要强调函数名称的时候使用，因此它还有一个名字叫做**匿名函数。它的结构只包含四部分，即lambda 关键字、需要接收的参数、一个冒号和对接收参数的处理**，并且它会把处理结果自动返回。

因此在 sorted() 函数中通过 lambda 表达式实现按关键字排序，会比使用自定义函数排序更加简洁，所以当你遇到只有一行语句的函数场景时，可以考虑使用 lambda 表达式替代自定义函数的定义和调用。

字典类型的排序方式

除了列表 + 元组的复合类型外，我们经常还需要对字典类型进行排序，**字典类型包含键和值，所以排序的时候可以基于键来排序，也可以基于值来排序。**

我们还是以学生信息为例。我把学生的姓名定义为字典的键，把学号定义为字典的值。接下来我们看看 sorted() 是怎么对字典的键和值进行排序的，实现排序的代码如下：

复制代码

```
student_dict1 = {'Jerry':'1003',
                 'Tom':'1005',
                 'Beta':'2001',
                 'Shuke':'2003'
}

# 输出字典的键和值
print(student_dict1.items())
# dict_items([('Jerry', '1003'), ('Tom', '1005'), ('Beta', '2001'), ('Shuke',

# 按照字典的键排序
print(sorted(student_dict1.items(), key=lambda d: d[0]))
# [('Beta', '2001'), ('Jerry', '1003'), ('Shuke', '2003'), ('Tom', '1005')]

# 按照字典的值排序
result = sorted(student_dict1.items(), key=lambda d: d[1])
print(result)
[('Jerry', '1003'), ('Tom', '1005'), ('Beta', '2001'), ('Shuke', '2003')]
# 将结果转换为字典
print(dict(result))
# {'Jerry': '1003', 'Tom': '1005', 'Beta': '2001', 'Shuke': '2003'}
```

在这段代码中，我实现了基于字典的键和字典的值进行排序的功能。由于字典是使用键值对的形式存储数据的，所以我先通过字典自带的函数 items()，把键值对的形式转换成列表 + 元组的形式。经过转换以后，字典的键就变成了元组的第一个参数，值就变成了元组的第二个参数。

因此在代码中，我把参数 key 的值设置为“lambda d: d[0]”，这样就取得了元组的第一个元素，排序之后就实现了按照字典的键进行排序的需求。相应的，使用“lambda d: d[1]”，可以取得元组第二个元素的值，就能实现按照字典的值进行排序的需求。

由于排序后的数据类型已经变成了列表 + 元组形式。所以我们在代码最后，可以通过 dict() 函数把排序的结果再转换为字典，和排序前的数据类型保持一致。

这就是通过 sorted() 函数对字典进行排序的解决方法，你在工作中或许还会遇到和这节课不一样的数据类型，但是它们的解决思路是相通的。

1. 如果能转换成列表，可以采用更改 lambda 下标的方式，实现对指定字段的排序。
2. 如果不能转换成列表，可以尝试将复杂的类型中，不需要进行排序的部分进行删减，简化成列表或字典类型，这样就也能使用 sorted 实现数据的排序功能了。

小结

最后，我来给你总结一下这节课的核心内容。我们通过 sorted() 函数实现了列表、列表 + 元组、字典类型的排序，通过排序后的结果，你可以快速提取 TopN，也可以利用你学习过的其他语言的算法实现更复杂的查找。

在 sorted() 函数中，我们通过编写 key 参数的值，使用了 lambda 表达式替代了简单的函数，让我们的程序更加简洁。当你在日后的工作场景中也出现了只有单个语句的函数时，也可以考虑采用 lambda 表达式替代自定义函数，增加代码的可读性。

正是因为 sorted() 功能的强大，我们在 python 中实现排序几乎不需要通过自行编写代码来实现排序，不过你在进行海量数据的时候，我还想给你提供两条有用的建议。

第一，如果对包含海量数据的列表进行排序时，建议将列表通过 tuple() 函数转换为元组，能够让查找效率有较大提升。

第二，在 Python 标准库 collections 库中提供了 OrderedDict 扩展数据类型，它的特点是对 OrderedDict 数据类型进行赋值时，会自动进行排序。当你需要一个有序字典时，可以考虑选择 OrderedDict 作为数据存储的类型，从而避免手工对数据排序。

我也把这节课的相关代码放在了 GitHub 上，你可以去学习查找。

思考题

在最后，我想给你留一道开放性的问题。为什么 Python 语言不将所有的数据类型都设计成默认有序的，即存入数据时自动进行排序？欢迎你说出自己的理由。

欢迎把你对问题的思考和想法分享在留言区，我们一起交流讨论。如果课程帮你解决了一些工作上的问题，也欢迎你把课程分享给你的朋友、同事。

提建议

© 版权归极客邦科技所有，未经许可不得传播售卖。 页面已增加防盗追踪，如有侵权极客邦将依法追究其法律责任。

上一篇 09 | 扩展搜索：如何快速找到想要的文件?

下一篇 11 | 通过程序并行计算，避免CPU资源浪费

精选留言 (1)

写留言

Soul of the Drago...

2021-03-04

关于思考题，我猜可以是照顾到用户的实际需求吧，并不是什么情况下都需要对数据进行排序的，只要能实现排序的功能，用户可以自行对数据进行排序。

展开

作者回复: 照顾到用户的实际需求是一方面,这方面你的理解是对的;还有一方面要考虑排序对计算机资源的开销,随着数据增加,排序的时间开销也会有相应增加(增加多少要根据具体算法和数据长度评估排序时间复杂度),所以在不需要排序时,我们还是需要一些能替我们存储数据,但是不需要自动排序的数据类型的